Content-Type: text/html

< draft-ietf-aqm-fq-implementation-02.txt	draft-ietf-aqm-fq-implementation-04.txt >

< draft-ietf-aqm-fq-implementation-02.txt

Active Queue Management F. Baker
Internet-Draft R. Pan
Intended status: Informational Cisco Systems
Expires: November 2, 2015 May 1, 2015

On Queuing, Marking, and Dropping
draft-ietf-aqm-fq-implementation-02

Abstract

This note discusses implementation strategies for coupled queuing and mark/drop algorithms.

Status of This Memo

Copyright Notice

draft-ietf-aqm-fq-implementation-04.txt >

Active Queue Management F. Baker
Internet-Draft R. Pan
Intended status: Informational Cisco Systems
Expires: April 24, 2016 October 22, 2015

On Queuing, Marking, and Dropping
draft-ietf-aqm-fq-implementation-04

Abstract

This note discusses queuing and marking/dropping algorithms. While these algorithms may be implemented in a coupled manner, this note argues that specifications, measurements, and comparisons should decouple the different algorithms and their contributions to system behavior.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

skipping to change at page 2, line 26 | skipping to change at page 2, line 26

1. Introduction

In the discussion of Active Queue Management, there has been

1. Introduction

In the discussion of Active Queue Management, there has been

skipping to change at page 5, line 26 | skipping to change at page 5, line 26

2.1.3. GPS Comparisons: unit of measurement

And finally, there is the question of what is measured for rate. If the only objective is to force packet streams to not dominate each other, it is sufficient to count packets. However, if the issue is the bit rate of an SLA, one must consider the sizes of the packets (the aggregate throughput of a flow, measured in bits or bytes). And if predictable unfairness is a consideration, the value must be weighted accordingly.

Briscoe discusses measurement in his paper on Byte and Packet Congestion Notification [RFC7141].

2.2. GPS Approximations

Carrying the matter further, a queuing algorithm may also be termed "Work Conserving" or "Non Work Conserving". A "work conserving" algorithm, by definition, is either empty, in which case no attempt is being made to dequeue data from it, or contains something, in which case it continuously tries to empty the queue. A work conserving queue that contains queued data, at an interface with a given rate, will deliver data at that rate until it empties. A non-work-conserving queue might stop delivering even though it still contains data. A common reason for doing this is to impose an artificial upper bound on a class of traffic that is lower than the rate of the underlying physical interface.

2.1.3. GPS Comparisons: unit of measurement

And finally, there is the question of what is measured for rate. If the only objective is to force packet streams to not dominate each other, it is sufficient to count packets. However, if the issue is the bit rate of an SLA, one must consider the sizes of the packets (the aggregate throughput of a flow, measured in bits or bytes). And if predictable unfairness is a consideration, the value must be weighted accordingly.

[RFC7141] discusses measurement.

2.2. GPS Approximations

Carrying the matter further, a queuing algorithm may also be termed "Work Conserving" or "Non Work Conserving". A queue in a "work conserving" algorithm, by definition, is either empty, in which case no attempt is being made to dequeue data from it, or contains something, in which case the algorithm continuously tries to empty the queue. A work conserving queue that contains queued data, at an interface with a given rate, will deliver data at that rate until it empties. A non-work-conserving queue might stop delivering even though it still contains data. A common reason for doing this is to impose an artificial upper bound on a class of traffic that is lower than the rate of the underlying physical interface.

```
2.2.1.  Definition of a queuing algorithm

   In the discussion following, we assume a basic definition of a
   queuing algorithm.  A queuing algorithm has, at minimum:

   o  Some form of internal storage for the elements kept in the queue,

   o  If it has multiple internal classifications,
      *  a method for classifying elements,

      *  additional storage for the classifier and implied classes,

   o  potentially, a method for creating the queue,

   o  potentially, a method for destroying the queue,

   o  a method, called "enqueue", for placing packets into the queue or
      queuing system

   o  a method, called "dequeue", for removing packets from the queue or
      queuing system

   There may also be other information or methods, such as the ability
   to inspect the queue.  It also often has inspectable external
   attributes, such as the total volume of packets or bytes in queue,
   and may have limit thresholds, such as a maximum number of packets or
   bytes the queue might hold.

   For example, a simple FIFO queue has a linear data structure,
   enqueues packets at the tail, and dequeues packets from the head.  It
   might have a maximum queue depth and a current queue depth,
   maintained in packets or bytes.

2.2.2.  Round Robin Models

   One class of implementation approaches, generically referred to as
   "Weighted Round Robin", implements the structure of the queue as an
   array or ring of sub-queues associated with flows, for whatever
   definition of a flow is important.

   On enqueue, the enqueue function classifies a packet and places it
   into a simple FIFO sub-queue.

   On dequeue, the sub-queues are searched in round-robin order, and
   when a sub-queue is identified that contains data, removes a
   specified quantum of data from it.  That quantum is at minimum a
   packet, but it may be more.  If the system is intended to maintain a
   byte rate, there will be memory between searches of the excess
   previously dequeued.

                            +-+
                          +>|1|
                          | +-+
                          |  |
                          | +-+               +-+
                          | |1|             +>|3|
                          | +-+             | +-+
                          |  |              |  |
                          | +-+      +-+    | +-+
```

```
2.2.1.  Definition of a queuing algorithm

   In the discussion following, we assume a basic definition of a
   queuing algorithm.  A queuing algorithm has, at minimum:

   o  Some form of internal storage for the elements kept in the queue,

   o  If it has multiple internal classifications,

      *  a method for classifying elements,
      *  additional storage for the classifier and implied classes,

   o  potentially, a method for creating the queue,

   o  potentially, a method for destroying the queue,

   o  an enqueuing method, for placing packets into the queue or queuing
      system

   o  a dequeuing method, for removing packets from the queue or queuing
      system

   There may also be other information or methods, such as the ability
   to inspect the queue.  It also often has inspectable external
   attributes, such as the total volume of packets or bytes in queue,
   and may have limit thresholds, such as a maximum number of packets or
   bytes the queue might hold.

   For example, a simple FIFO queue has a linear data structure,
   enqueues packets at the tail, and dequeues packets from the head.  It
   might have a maximum queue depth and a current queue depth,
   maintained in packets or bytes.

2.2.2.  Round Robin Models

   One class of implementation approaches, generically referred to as
   "Weighted Round Robin" (WRR), implements the structure of the queue
   as an array or ring of sub-queues associated with flows, for whatever
   definition of a flow is important.

   The arriving packet must, of course, first be classified.  If a hash
   is used as a classifier, the modulus of the hash might be used as an
   array index, selecting the sub-queue that the packet will go into.
   One can imagine other classifiers, such as using a Differentiated
   Services Code Point (DSCP) value as an index into an array containing
   the queue number for a flow, or more complex access list
   implementations.

   In any event, a sub-queue contains the traffic for a flow, and data
   is sent from each sub-queue in succession.

   On enqueue, the enqueue method places a classified packet into a
   simple FIFO sub-queue.

   On dequeue, the sub-queues are searched in round-robin order, and
   when a sub-queue is identified that contains data, the dequeue method
   removes a specified quantum of data from it.  That quantum is at
   minimum a packet, but it may be more.  If the system is intended to
   maintain a byte rate, there will be memory between searches of the
   excess previously dequeued.

                            +-+
                          +>|1|
                          | +-+
                          |  |
                          | +-+               +-+
                          | |1|             +>|3|
                          | +-+             | +-+
                          |  |              |  |
                          | +-+      +-+    | +-+
```

skipping to change at *page 7, line 25* **skipping to change at *page 7, line 27***

```
                          | +-+     | +-+     | +-+
                          |  A      |  A      |  A
                          |  |      |  |      |  |
                         ++--++    ++--++    ++--++
                      +->| Q  |-->| Q  |-->| Q  |--+
                      |  +----+   +----+   +----+  |
                      +----------------------------+

                       Figure 1: Round Robin Queues

   If a hash is used as a classifier, the modulus of the hash might be
   used as an array index, selecting the sub-queue that the packet will
   go into.  One can imagine other classifiers, such as using a
   Differentiated Services Code Point (DSCP) value as an index into an
   array containing the queue number for a flow, or more complex access
   list implementations.

   In any event, a sub-queue contains the traffic for a flow, and data
   is sent from each sub-queue in succession.

2.2.3.  Calendar Queue Models

   Another class of implementation approaches, generically referred to
   as "Weighted Fair Queues" or "Calendar Queue Implementations",
```

```
                          | +-+     | +-+     | +-+
                          |  A      |  A      |  A
                          |  |      |  |      |  |
                         ++--++    ++--++    ++--++
                      +->| Q  |-->| Q  |-->| Q  |--+
                      |  +----+   +----+   +----+  |
                      +----------------------------+

                       Figure 1: Round Robin Queues

2.2.3.  Calendar Queue Models

   Another class of implementation approaches, generically referred
   Calendar Queue Implementations [CalendarQueue], implements the
```

implements the structure of the queue as an array or ring of sub-queues (often called "buckets") associated with time or sequence; Each bucket contains the set of packets, which may be null, intended to be sent at a certain time or following the emptying of the previous bucket. The queue structure includes a look-aside table that indicates the current depth (which is to say, the next bucket) of any given class of traffic, which might similarly be identified using a hash, a DSCP, an access list, or any other classifier. Conceptually, the queues each contain zero or more packets from each class of traffic. One is the queue being emptied "now"; the rest are associated with some time or sequence in the future.

On enqueue, the enqueue function classifies a packet and determines the current depth of that class, with a view to scheduling it for transmission at some time or sequence in the future. If the unit of scheduling is a packet and the queuing quantum is one packet per sub-queue, a burst of packets arrives in a given flow, and at the start the flow has no queued data, the first packet goes into the "next" queue, the second into its successor, and so on; if there was some data in the class, the first packet in the burst would go into the bucket pointed to by the look-aside table. If the unit of scheduling is time, the explanation in Section 2.2.5 might be simplest to follow, but the bucket selected will be the bucket corresponding to a given transmission time in the future. A necessary side-effect, memory being finite, is that there exist a finite number of "future" buckets. If enough traffic arrives to cause a class to wrap, one is forced to drop something (tail-drop).

On dequeue, the buckets are searched at their stated times or in their stated sequence, and when a bucket is identified that contains data, removes a specified quantum of data from it and, by extension, from the associated traffic classes. A single bucket might contain data from a number of classes simultaneously.

```
                          +-+
                        +>|1|
                        | +-+
                        |  |
                        | +-+     +-+
                        | |2|   +>|2|
                        | +-+   | +-+
                        |  |    |  |
                        | +-+   | +-+     +-+
```

skipping to change at *page 9, line 13*

end case develops: If the system is draining a given sub-queue, and the time of the next sub-queue arrives, what should the system do? One potentially valid line of reasoning would have it continue delivering the data in the present queue, on the assumption that it will likely trade off for time in the next. Another potentially valid line of reasoning would have it discard any waiting data in the present queue and move to the next.

2.2.4. Work Conserving Models and Stochastic Fairness Queuing

McKenney's Stochastic Fairness Queuing [SFQ] is an example of a work conserving algorithm. This algorithm measures packets, and considers a "flow" to be an equivalence class of traffic defined by a hashing algorithm over the source and destination IPv4 addresses. As packets arrive, the enqueue function performs the indicated hash and places the packet into the indicated sub-queue. The dequeue function operates as described in Section 2.2.2; sub-queues are inspected in round-robin sequence, and if they contain one or more packets, a packet is removed.

Shreedhar's Deficit Round Robin [DRR] model modifies the quanta to bytes, and deals with variable length packets. A sub-queue descriptor contains a waiting quantum (the amount intended to be dequeued on the previous dequeue attempt that was not satisfied), a per-round quantum (the sub-queue is intended to dequeue a certain number of bytes each round), and a maximum to permit (some multiple of the MTU). In each dequeue attempt, the dequeue method sets the waiting quantum to the smaller of the maximum quantum and the sum of the waiting and incremental quantum. It then dequeues up to the waiting quantum, in bytes, of packets in the queue, and reduces the waiting quantum by the number of bytes dequeued. Since packets will not normally be exactly the size of the quantum, some dequeue attempts will dequeue more than others, but they will over time average the incremental quantum per round if there is data present.

McKenny or Shreedhar's models could be implemented as described in Section 2.2.3. The weakness of a WRR approach is the search time expended when the queuing system is relatively empty, which the calendar queue model obviates.

2.2.5. Non Work Conserving Models and Virtual Clock

Zhang's Virtual Clock [VirtualClock] is an example of a non-work-conserving algorithm. It is trivially implemented as described in Section 2.2.3. It associates buckets with intervals in time, with durations on the order of microseconds to tens of milliseconds. Each flow is assigned a rate in bytes per interval. The flow entry maintains a point in time the "next" packet in the flow should be scheduled.

On enqueue, the method determines whether the "next schedule" time is "in the past"; if so, the packet is scheduled "now", and if not, the packet is scheduled at that time. It then calculates the new "next

---

structure of the queue as an array or ring of sub-queues (often called "buckets") associated with time or sequence; Each bucket contains the set of packets, which may be null, intended to be sent at a certain time or following the emptying of the previous bucket. The queue structure includes a look-aside table that indicates the current depth (which is to say, the next bucket) of any given class of traffic, which might similarly be identified using a hash, a DSCP, an access list, or any other classifier. Conceptually, the queues each contain zero or more packets from each class of traffic. One is the queue being emptied "now"; the rest are associated with some time or sequence in the future.

On enqueue, the enqueue method, considering a classified packet, determines the current depth of that class with a view to scheduling it for transmission at some time or sequence in the future. If the unit of scheduling is a packet and the queuing quantum is one packet per sub-queue, a burst of packets arrives in a given flow, and at the start the flow has no queued data, the first packet goes into the "next" queue, the second into its successor, and so on; if there was some data in the class, the first packet in the burst would go into the bucket pointed to by the look-aside table. If the unit of scheduling is time, the explanation in Section 2.2.5 might be simplest to follow, but the bucket selected will be the bucket corresponding to a given transmission time in the future. A necessary side-effect, memory being finite, is that there exist a finite number of "future" buckets. If enough traffic arrives to cause a class to wrap, one is forced to drop something (tail-drop).

On dequeue, the buckets are searched at their stated times or in their stated sequence, and when a bucket is identified that contains data, the dequeue method removes a specified quantum of data from it and, by extension, from the associated traffic classes. A single bucket might contain data from a number of classes simultaneously.

```
                          +-+
                        +>|1|
                        | +-+
                        |  |
                        | +-+     +-+
                        | |2|   +>|2|
                        | +-+   | +-+
                        |  |    |  |
                        | +-+   | +-+     +-+
```

skipping to change at *page 9, line 7*

end case develops: If the system is draining a given sub-queue, and the time of the next sub-queue arrives, what should the system do? One potentially valid line of reasoning would have it continue delivering the data in the present queue, on the assumption that it will likely trade off for time in the next. Another potentially valid line of reasoning would have it discard any waiting data in the present queue and move to the next.

2.2.4. Work Conserving Models and Stochastic Fairness Queuing

Stochastic Fairness Queuing [SFQ] is an example of a work conserving algorithm. This algorithm measures packets, and considers a "flow" to be an equivalence class of traffic defined by a hashing algorithm over the source and destination IPv4 addresses. As packets arrive, the enqueue method performs the indicated hash and places the packet into the indicated sub-queue. The dequeue method operates as described in Section 2.2.2; sub-queues are inspected in round-robin sequence, and if they contain one or more packets, a packet is removed.

Deficit Round Robin [DRR] model modifies the quanta to bytes, and deals with variable length packets. A sub-queue descriptor contains a waiting quantum (the amount intended to be dequeued on the previous dequeue attempt that was not satisfied), a per-round quantum (the sub-queue is intended to dequeue a certain number of bytes each round), and a maximum to permit (some multiple of the MTU). In each dequeue attempt, the dequeue method sets the waiting quantum to the smaller of the maximum quantum and the sum of the waiting and incremental quantum. It then dequeues up to the waiting quantum, in bytes, of packets in the queue, and reduces the waiting quantum by the number of bytes dequeued. Since packets will not normally be exactly the size of the quantum, some dequeue attempts will dequeue more than others, but they will over time average the incremental quantum per round if there is data present.

[SFQ] and [DRR] could be implemented as described in Section 2.2.3. The weakness of a WRR approach is the search time expended when the queuing system is relatively empty or the overhead of obviating that issue, which the calendar queue model also obviates.

2.2.5. Non Work Conserving Models and Virtual Clock

Virtual Clock [VirtualClock] is an example of a non-work-conserving algorithm. It is trivially implemented as described in Section 2.2.3. It associates buckets with intervals in time, with durations on the order of microseconds to tens of milliseconds. Each flow is assigned a rate in bytes per interval. The flow entry maintains a point in time the "next" packet in the flow should be scheduled.

On enqueue, the method determines whether the "next schedule" time is "in the past"; if so, the packet is scheduled "now", and if not, the packet is scheduled at that time. It then calculates the new "next

draft-ietf-aqm-fq-implementation-02.txt	draft-ietf-aq...
schedule" time, as the current "next schedule" time plus the length	schedule" time, as the current "next schedule" time plus the length
skipping to change at *page 12, line 32*	**skipping to change at *page 12, line 27***
To summarize, in Section 2, implementation approaches for several classes of queuing algorithms were explored. Queuing algorithms such as SFQ, Virtual Clock, and FlowQueue-Codel [I-D.ietf-aqm-fq-codel] have value in the network, in that they delay packets to enforce a rate upper bound or to permit competing flows to compete more effectively. ECN Marking and loss are also useful signals if used in a manner that enhances TCP/SCTP operation or restrains unmanaged UDP data flows.	To summarize, in Section 2, implementation approaches for several classes of queuing algorithms were explored. Queuing algorithms such as SFQ, Virtual Clock, and FlowQueue-Codel [I-D.ietf-aqm-fq-codel] have value in the network, in that they delay packets to enforce a rate upper bound or to permit competing flows to compete more effectively. ECN Marking and loss are also useful signals if used in a manner that enhances TCP/SCTP operation or restrains unmanaged UDP data flows.
Conceptually, queuing algorithms and a mark/drop algorithms operate in series, as discussed in Section 3, not as a single algorithm. The observed effects differ: defensive loss protects the intermediate system and provides a signal, AQM mark/drop works to reduce mean latency, and the scheduling of flows works to modify flow interleave and acknowledgement pacing. Certain features like flow isolation are provided by fair queuing related designs, but are not the effect of the mark/drop algorithm.	Conceptually, queuing algorithms and mark/drop algorithms operate in series, as discussed in Section 3, not as a single algorithm. The observed effects differ: defensive loss protects the intermediate system and provides a signal, AQM mark/drop works to reduce mean latency, and the scheduling of flows works to modify flow interleave and acknowledgement pacing. Certain features like flow isolation are provided by fair queuing related designs, but are not the effect of the mark/drop algorithm.
There is value in implementing and coupling the operation of both queuing algorithms and queue management algorithms, and there is definitely interesting research in this area, but specifications,	There is value in implementing and coupling the operation of both queuing algorithms and queue management algorithms, and there is definitely interesting research in this area, but specifications,
skipping to change at *page 13, line 23*	**skipping to change at *page 13, line 18***
discussions in AQM, in which some have pushed an algorithm the compare to AQM marking and dropping algorithms, but which includes Flow Queuing.	discussions in AQM, in which some have pushed an algorithm the compare to AQM marking and dropping algorithms, but which includes Flow Queuing.
8. References	8. References
8.1. Normative References	8.1. Normative References
[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.	[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <http://www.rfc-editor.org/info/rfc2475>.
8.2. Informative References	8.2. Informative References
	[CalendarQueue] "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem", Communications of the ACM 1988, October 1988, <http://www.cs.ucla.edu/~lixia/papers/90sigcomm.pdf>.
[DRR] Microsoft Corporation and Washington University in St. Louis, "Efficient fair queueing using deficit round robin", ACM SIGCOMM 1995, October 1995, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=502236>.	[DRR] Microsoft Corporation and Washington University in St. Louis, "Efficient fair queueing using deficit round robin", ACM SIGCOMM 1995, October 1995, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=502236>.
[GPS] Xerox PARC, University of California, Berkeley, and Xerox PARC, "Analysis and simulation of a fair queueing algorithm", ACM SIGCOMM 1989, September 1989, <http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/89/fq.pdf>.	[GPS] Xerox PARC, University of California, Berkeley, and Xerox PARC, "Analysis and simulation of a fair queueing algorithm", ACM SIGCOMM 1989, September 1989, <http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/89/fq.pdf>.
[I-D.ietf-aqm-codel] Nichols, K., Jacobson, V., McGregor, A., and J. Jana, "Controlled Delay Active Queue Management", draft-ietf-aqm-codel-01 (work in progress), April 2015.	[I-D.ietf-aqm-codel] Nichols, K., Jacobson, V., McGregor, A., and J. Jana, "Controlled Delay Active Queue Management", draft-ietf-aqm-codel-01 (work in progress), April 2015.
[I-D.ietf-aqm-fq-codel] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "FlowQueue-Codel", draft-ietf-aqm-fq-codel-00 (work in progress), January 2015.	[I-D.ietf-aqm-fq-codel] Hoeiland-Joergensen, T., McKenney, P., dave.taht@gmail.com, d., Gettys, J., and E. Dumazet, "FlowQueue-Codel", draft-ietf-aqm-fq-codel-02 (work in progress), October 2015.
[I-D.ietf-aqm-pie] Pan, R., Natarajan, P., Baker, F., and G. White, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-ietf-aqm-pie-01 (work in progress), March 2015.	[I-D.ietf-aqm-pie] Pan, R., Natarajan, P., and F. Baker, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-ietf-aqm-pie-02 (work in progress), August 2015.
[NoFair] British Telecom, "Flow rate fairness: dismantling a religion", ACM SIGCOMM 2007, April 2007, <http://dl.acm.org/citation.cfm?id=1232926>.	[NoFair] British Telecom, "Flow rate fairness: dismantling a religion", ACM SIGCOMM 2007, April 2007, <http://dl.acm.org/citation.cfm?id=1232926>.
[RFC0970] Nagle, J., "On packet switches with infinite storage", RFC 970, December 1985.	[RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <http://www.rfc-editor.org/info/rfc970>.
[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.	[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998, <http://www.rfc-editor.org/info/rfc2309>.
[RFC2990] Huston, G., "Next Steps for the IP QoS Architecture", RFC 2990, November 2000.	[RFC2990] Huston, G., "Next Steps for the IP QoS Architecture", RFC 2990, DOI 10.17487/RFC2990, November 2000, <http://www.rfc-editor.org/info/rfc2990>.
[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition	[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition

of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.	of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.
[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.	[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, DOI 10.17487/RFC3390, October 2002, <http://www.rfc-editor.org/info/rfc3390>.
[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.	[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <http://www.rfc-editor.org/info/rfc5690>.
[RFC6057] Bastian, C., Klieber, T., Livingood, J., Mills, J., and R. Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, December 2010.	[RFC6057] Bastian, C., Klieber, T., Livingood, J., Mills, J., and R. Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, DOI 10.17487/RFC6057, December 2010, <http://www.rfc-editor.org/info/rfc6057>.
[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.	[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <http://www.rfc-editor.org/info/rfc6679>.
[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.	[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <http://www.rfc-editor.org/info/rfc6928>.
[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, February 2014.	[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <http://www.rfc-editor.org/info/rfc7141>.
[SFQ] SRI International, "Stochastic Fairness Queuing", IEEE Infocom 1990, June 1990, <http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>.	[SFQ] SRI International, "Stochastic Fairness Queuing", IEEE Infocom 1990, June 1990, <http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>.
[VirtualClock] Xerox PARC, "Virtual Clock", ACM SIGCOMM 1990, September 1990, <http://www.cs.ucla.edu/~lixia/papers/90sigcomm.pdf>.	[VirtualClock] Xerox PARC, "Virtual Clock", ACM SIGCOMM 1990, September 1990, <http://www.cs.ucla.edu/~lixia/papers/90sigcomm.pdf>.

End of changes. 37 change blocks.

125 lines changed or deleted	***146 lines changed or added***

This html diff was produced by rfcdiff 1.42. The latest version is available from http://tools.ietf.org/tools/rfcdiff/